UM EECS 270 F22
Introduction to Logic Design

12. Two-Level Logic Minimization

Logic Synthesis: From Specs to Circuits

Synthesizing from some logic expression into a gate circuit (logic gate circuit)

- Implementation Styles
  - Random
  - Regular
- Optimization Criteria
  - Area (roughly number of transistors)
  - Delay (roughly number of gates on longest circuit path)
  - Testability
  - Power consumption
- 2-Level Synthesis
  - Classical (Exact)
  - Heuristic (Non-Exact)
- Multilevel Synthesis

traditional optimization criterion: is minimize area.
如何判断 area 的大小呢：去数我们有多少 transistor，即为我们可以用 transistor 的数量来 estimate area of the gate.
是 variable in our expression 的是 complemented 的也是 uncomplemented 的
Now, abstract from transistors to literals，其中 ones: every input to a gate goes to 2 transistors: pull up: pmos / pull down: nmos → So we can use how many literals we have in an expression as a proxy for how many transistors are in the gate → approxy how big the gate.

Some regularity / repetition
example: add and subtract circuit. Because it has some semantic meaning at some level, and there is some regularity in that structure.
Not just a bunch of gates connect randomly by wires

但是 gate 的 delay 也和其 width 相关，input 的数量。所以用 delay 来 approximate Area 相对比较困难。

assembly line to test them

when the circuit is operating, especially go to higher frequency

when the number of inputs becomes really large, these algorithms become unscalable.

means: gates circuits that have multiple levels of logic between inputs and output or between inputs and flip-flops that store memory (state)

Not just one way to synthesize

Logic Expressions ⇔ Circuits

Logic expressions and circuits have a one-to-one correspondence.

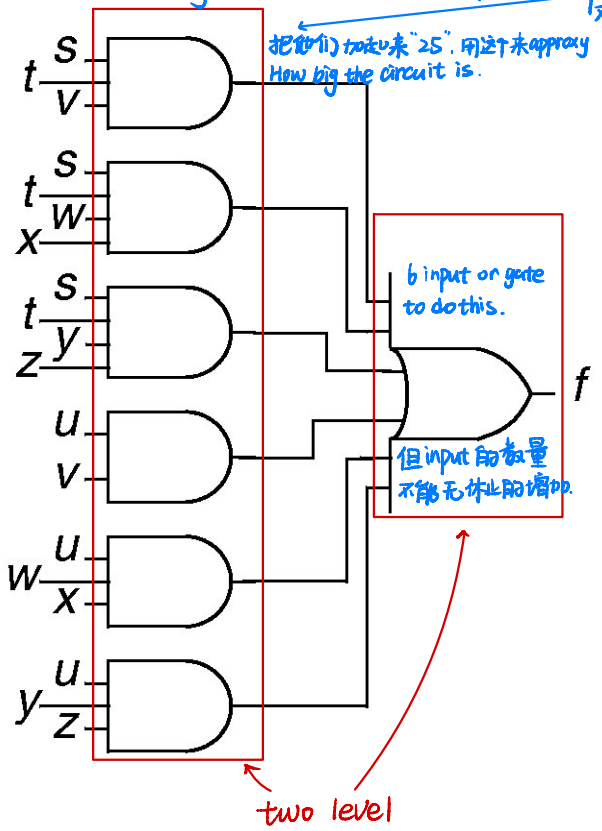
SOP

$$f = stv + stwx + styz + uv + uwx + uyz$$

what is the size of the circuit?

count how many literals are in the expression: 对于 and gate 有 1 个. 对于 or gate 有 6 个.

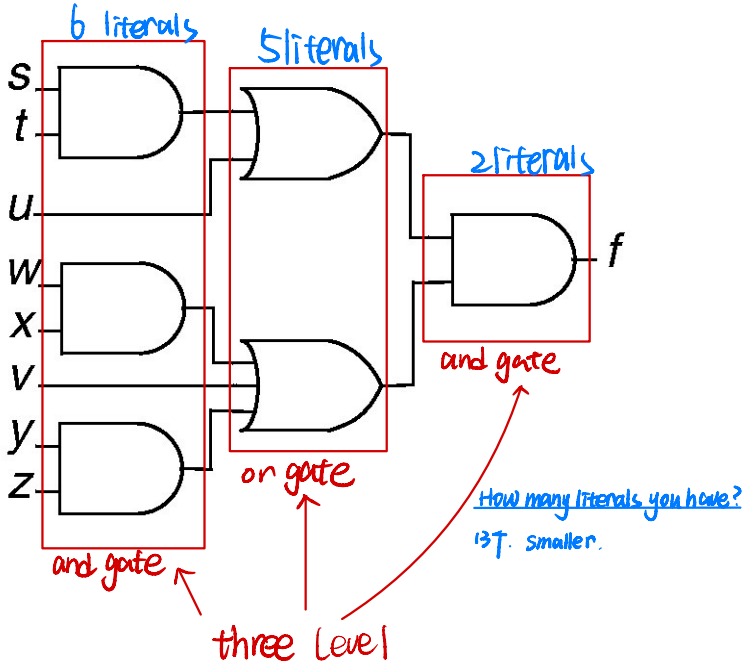
把他们加起来 "25". 用这个来approxy How big the circuit is.



it's not POS, because there exist product in sum. 不是SOP也不是POS

$$f = (st + u)(wx + yz + v)$$

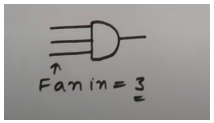
我们通过 factorization 得到这个.

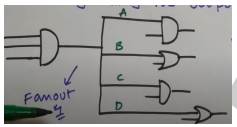


- Smaller circuit.
- it could also be faster. (因为 gate 的 max input 是 3)

所以这个 6o 左侧的 preferable

2-Level Circuits

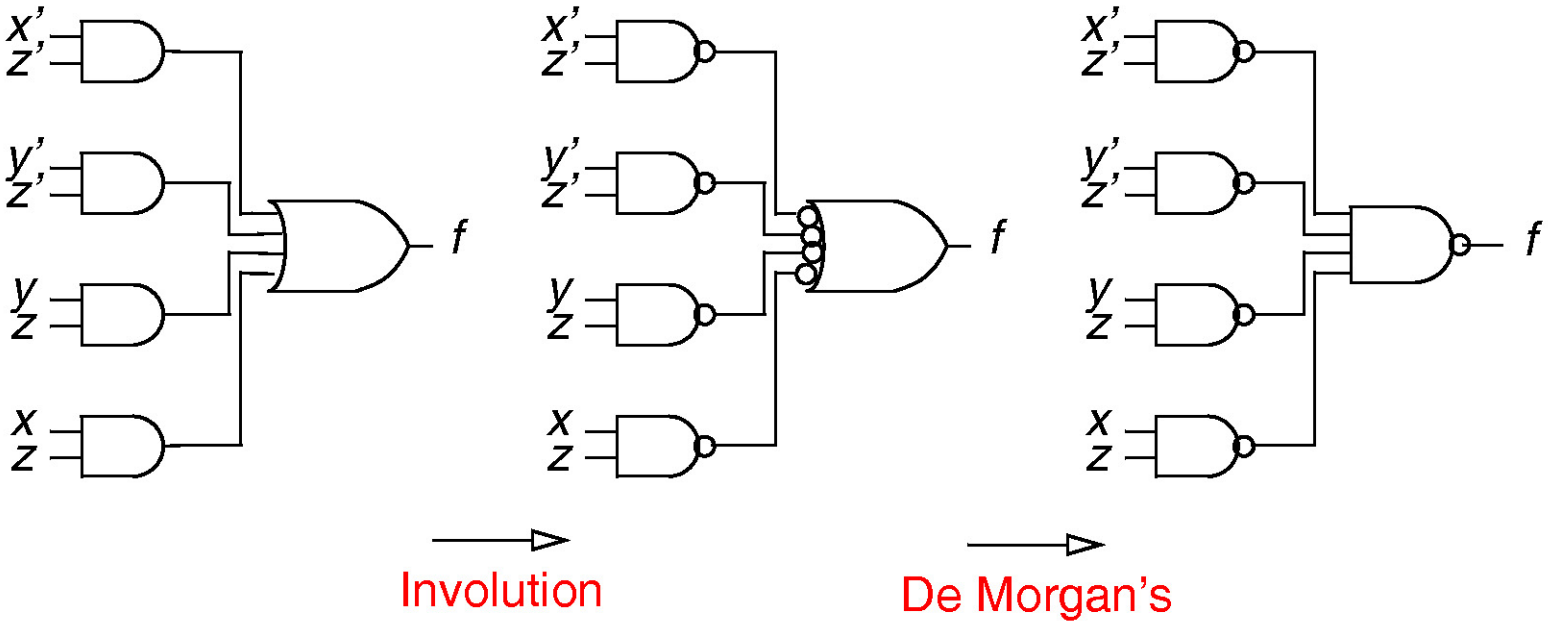




- Assumptions:
  - Inputs are available in both true and complemented forms (reasonable assumption: FF outputs are available in both phases)
  - Gates have no fan-in or fan-out restrictions (unreasonable assumption: typically FI < 4, FO < 6)
- Correspond to SOP or POS forms of a switching function:
  - SOP ⇔ AND/OR (NAND/NAND)
  - POS ⇔ OR/AND (NOR/NOR)
- # of 1st level AND (OR) gates = # of nontrivial product (sum) terms
- Fan-in of 2nd level OR (AND) gate = total # of product (sum) terms
- Usually implemented using regular structures:
  - Read-Only Memories (ROMs)
  - Programmable Logic Arrays (PLAs)

the number of inputs to a gate
Sometime we want to actually use the output of the gate at 1st level to go to 2 "or gate" in the output. higher the fan-out, the weaker the signal is. IE) if you're trying to drive many more things than just one gate.
they are the same

AND/OR ⇔ NAND/NAND



Terminology Review

- **Literal**: A variable or its complement
  - Ex: $X, Y, \overline{B}$
- **Product Term**: A single literal or a product of two or more literals
  - Ex: $XY, A\overline{B}C, \overline{W}X\overline{Y}Z$

that's a trivial productor it doesn't need a gate.

- **Sum of Products**: A sum of product terms
- **Minterm**: A product term where every variable of the function appears once as a literal (i.e., in complemented or uncomplemented form)
- **Canonical Sum**: A sum of minterms. Every function has a unique canonical sum (up to associativity of OR)

Canonical Sum Representations

- Canonical sum uniquely defines a function, but can be long. A **minterm list** is a list of the row numbers of minterms included in the canonical sum.
  - Also known as the **on-set**

These are the roads in the truth table for the function value must be one

- Three canonical ways of representing functions
  - truth table
  - canonical sum
  - minterm list

Row #	A	B	C	F
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

$$F = \sum_{A,B,C}(2,6,7)$$

Canonical Product Representation

- **Sum Term**: A single literal or a sum of two or more literals
- **Product of Sums**: A product of sum terms
- **Maxterm**: A sum term where every variable in the function appears once as a literal
- **Canonical Product**: A product of maxterms
- **Maxterm List**: A list of the truth table row numbers of the maxterms of a function
  - Also known as the **off-set**

Maxterm corresponding to 0

minterm : on-set
maxterm : off-set
don't cares : offset

Row #	A	B	C	F	Maxterms
0	0	0	0	0	A+B+C
1	0	0	1	1	A+B+$\bar{C}$
2	0	1	0	1	A+$\bar{B}$+C
3	0	1	1	1	A+$\bar{B}$+$\bar{C}$
4	1	0	0	1	$\bar{A}$+B+C
5	1	0	1	0	$\bar{A}$+B+$\bar{C}$
6	1	1	0	1	$\bar{A}$+$\bar{B}$+C
7	1	1	1	0	$\bar{A}$+$\bar{B}$+$\bar{C}$

$$F = (A+B+C)(\overline{A}+B+\overline{C})(\overline{A}+\overline{B}+\overline{C})$$

$$F = \Pi_{A,B,C}(0,5,7)$$

Think about how can we minimize

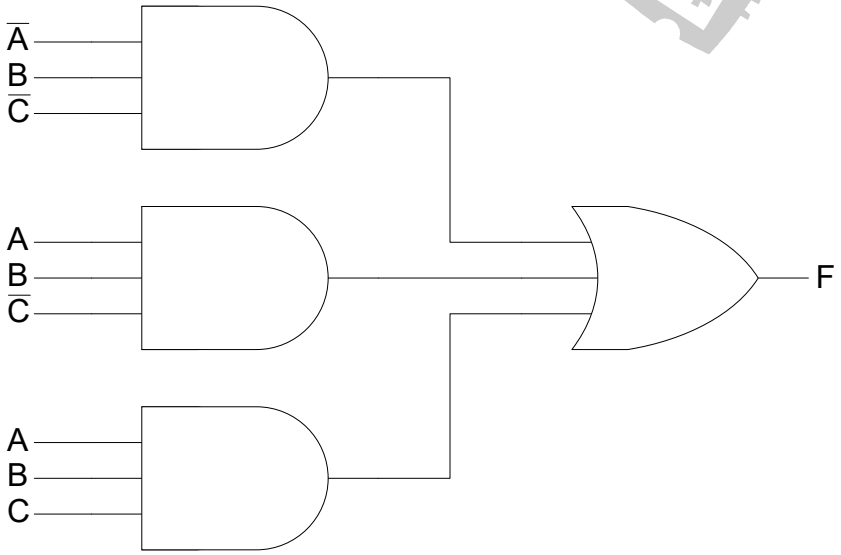
A	B	C	F	Minterms
0	0	0	0	$\bar{A}\bar{B}\bar{C}$
0	0	1	0	$\bar{A}\bar{B}C$
0	1	0	1	$\bar{A}B\bar{C}$
0	1	1	0	$\bar{A}BC$
1	0	0	0	$A\bar{B}\bar{C}$
1	0	1	0	$A\bar{B}C$
1	1	0	1	$AB\bar{C}$
1	1	1	1	ABC

能做什么，让这个变简单吗？(见下页)

Canonical Sum:

$$F = \bar{A}B\bar{C} + AB\bar{C} + ABC$$

Circuit:



Cost of implementation: **3 3-input ANDs (9 literals) + 1 3-input OR**

- Canonical sum is *not* (in general) the least-cost way to implement a function
- How to get smaller circuits?
  - One way: Switching algebra theorems!

Q: 我们依之什么来 minimize expression 呢?

We look for patterns where thing are adjacent

Q: terms adjecent 什么意思

they agree on everything except one variable

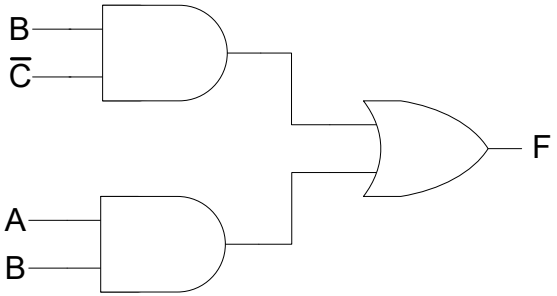
logical adjacency

variable that is compliment in one but not in other can be removed

$$F = \overline{A}B\overline{C} + AB\overline{C} + ABC$$

$$= (\overline{A}B\overline{C} + AB\overline{C}) + (AB\overline{C} + ABC) \quad \text{Idempotency}$$

$$= B\overline{C} + AB \quad \text{Combining}$$



Cost: 2 2-input ANDs (4 literals) + 1 2-input OR!

Applying theorems directly becomes more difficult as canonical sums get bigger…

$$\overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}C\overline{D} + \overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + \overline{A}BCD + A\overline{B}\overline{C}D + A\overline{B}C\overline{D} + AB\overline{C}\overline{D} + A$$

do the algebra graphically

- Another (perhaps easier) way: **Karnaugh maps**

Karnaugh Maps

- Karnaugh map (K-map) is a 2D representation of truth table that present a graphical way to find the minimal SOP (or POS) of a function
  - Each cell represents a minterm
  - Cells with a common edge are logically adjacent (i.e., they will combine)
  - K-map is actually a doughnut: outside edges connect

this called normal counting sequence

	A	B	C	F
0	0	0	0	m_0
1	0	0	1	m_1
2	0	1	0	m_2
3	0	1	1	m_3
4	1	0	0	m_4
5	1	0	1	m_5
6	1	1	0	m_6
7	1	1	1	m_7

idea:
- 右侧的每一个 square 对应左侧的每一行
- 右侧相邻的两个 square 是 adjacent 的关系

the number for the AB variables is not numerical

it's 0 1 3 2 ← called the great counting sequence.

2 bit changes, a bit changes, 1 bit changes

from one count to the next and only one bit changes

每次只改变一个东西，很容易看出 error

F, AB / C	00	01	11	10
0	m_0	m_2	m_6	m_4
1	m_1	m_3	m_7	m_5

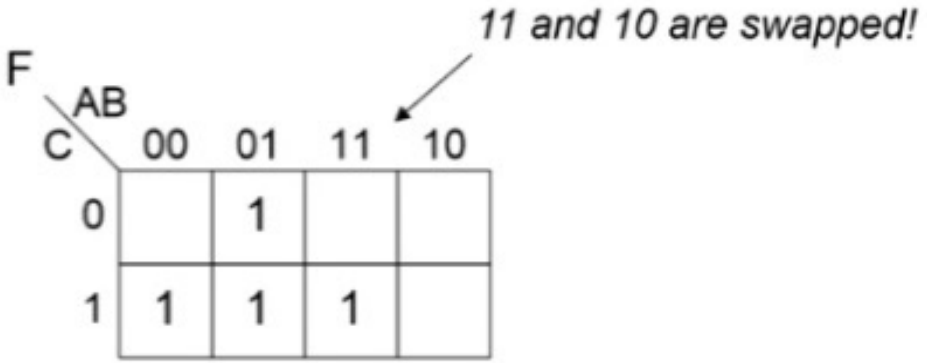
把这个表想像成 3D 圆柱体. m_5 和 m_1 也是 adjacent 的关系.

We'll omit 0s for clarity

Karnaugh Maps

- Karnaugh map (K-map) is a 2D representation of truth table that present a graphical way to find the minimal SOP (or POS) of a function
  - Each cell represents a minterm
  - Cells with a common edge are logically adjacent (i.e., they will combine)
  - K-map is actually a doughnut: outside edges connect

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

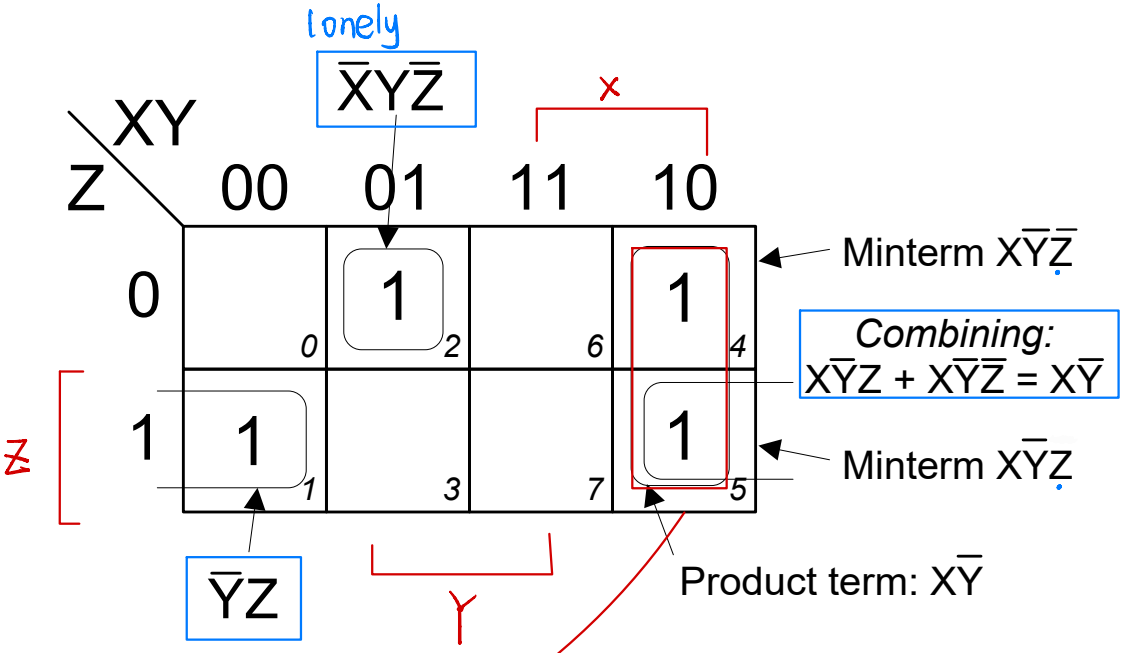


We'll omit 0s for clarity

- Key K-map property: *adjacent cells represent minterms that will combine!*

看每个 minterm 的 binary 的 expression.
如果超过一个 bit 不同，则 就不是 adjacent

$$F = \sum_{X,Y,Z}(1,2,4,5)$$



Reading product term:

X is 1 in both minterms: X

Y is 0 in both minterms: $\bar{Y}$

Z is 1 in one minterm and 0 in the other: Z does not appear in product term

look at the right-most column what we can say?
x is true, y is not true → $x\bar{y}$
we doing set algebra

Minimal SOP:

$$F = X\overline{Y} + \overline{Y}Z + \overline{X}\,\overline{Y}\,\overline{Z}$$

the cost is:
2+2+3+1x3 = 10

Four Variable K-map

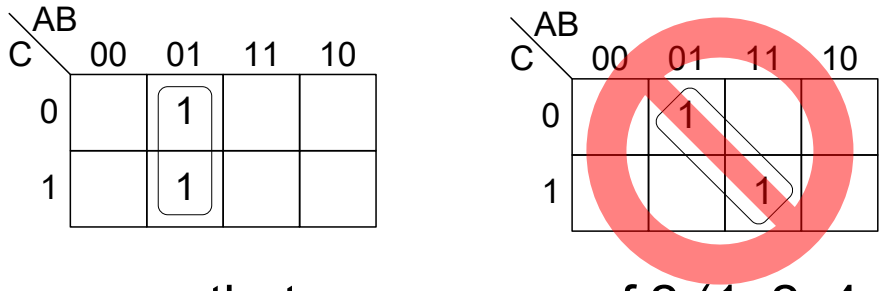
$$F = \sum_{W,X,Y,Z} (0,1,2,3,4,5,7,14,15)$$

F: YZ \ WX	00	01	11	10
00	m_0	m_4	m_{12}	m_8
01	m_1	m_5	m_{13}	m_9
11	m_3	m_7	m_{15}	m_{11}
10	m_2	m_6	m_{14}	m_{10}

F: YZ \ WX	00	01	11	10
00	1 (0)	1 (4)	(12)	(8)
01	1 (1)	1 (5)	(13)	(9)
11	1 (3)	1 (7)	1 (15)	(11)
10	1 (2)	(6)	1 (14)	(10)

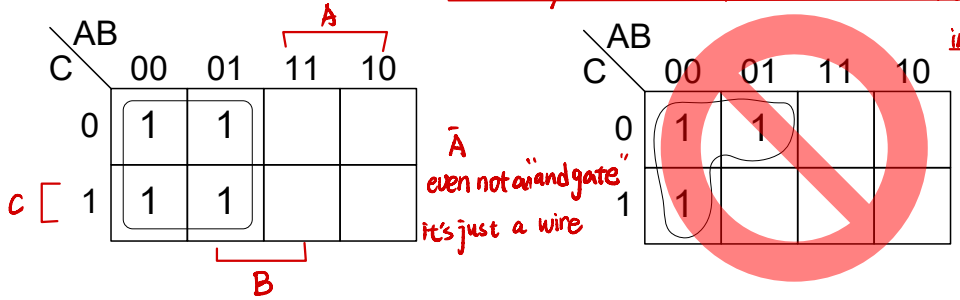
How do we find the product terms to include in the minimal SOP?

- K-map “rules”
  - Only circle adjacent cells (remember edges are adjacent!)

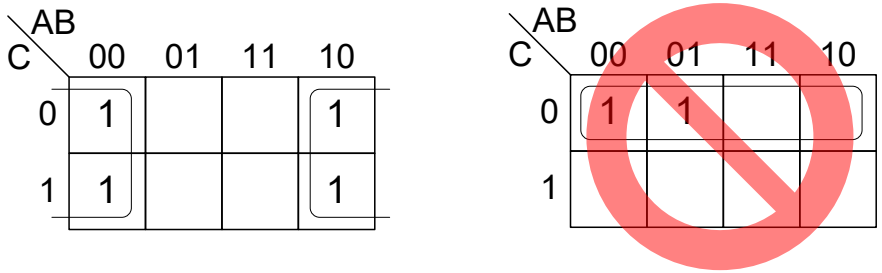


  - Only circle groups that are powers of 2 (1, 2, 4, 8, …)

It's not you can't circle it. But remember we're circling group of minterms that can be implemented as "an and gate"



  - Only circle 1-cells



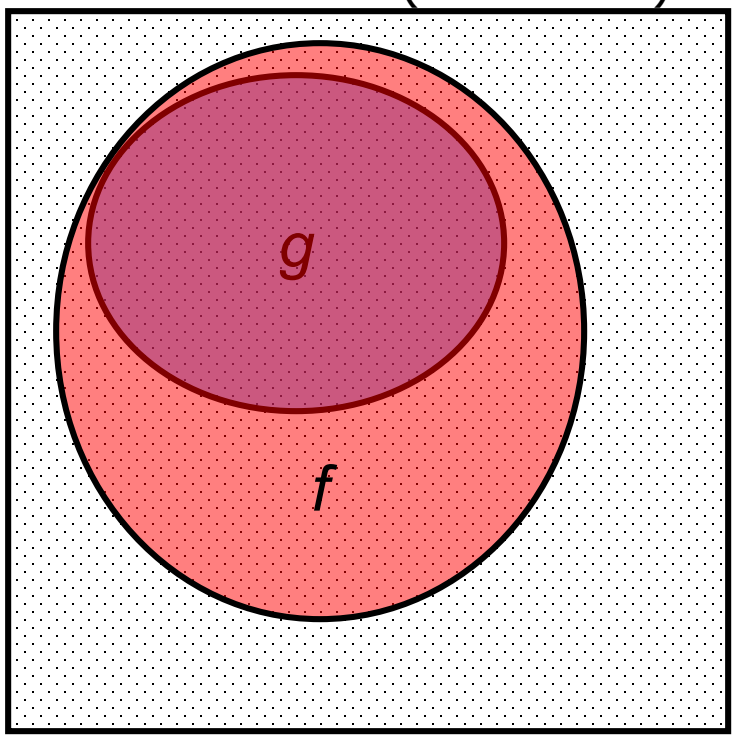
Covering

= input combinations for which *g* outputs 1

= input combinations for which *f* outputs 1

we can also say: g implies f $g \rightarrow f$

因为 g是对的 时候，f一定是对的



f* covers *g

f=1 whenever *g*=1

$$f \geq g$$

Implicants

= input combinations for which *g* outputs 1

= input combinations for which *f* outputs 1

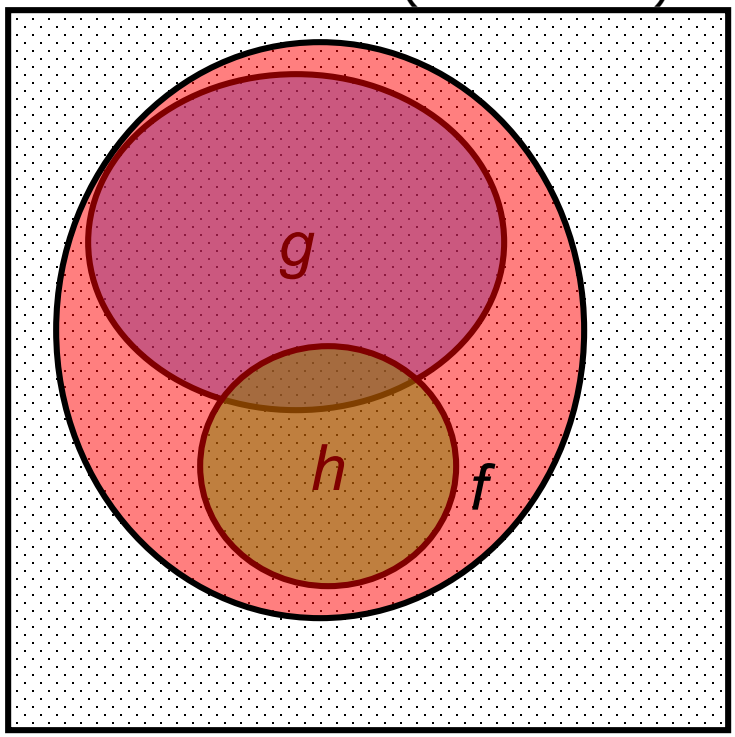
= input combinations for which *h* outputs 1

If *g* is a product term & $g \leq f$,
Then *g* is an **implicant** of *f*.

means it's an "and" literal x literal x ···

我们专门给他个名字.

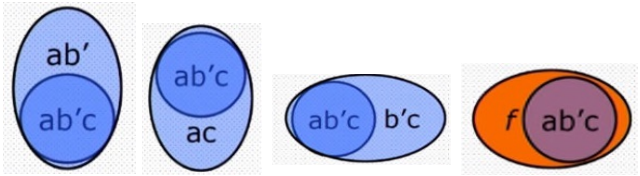
Space of all 2^n input combinations (minterms)



The key to minimization is finding the best implicants or the smallest and gates

Prime Implicants

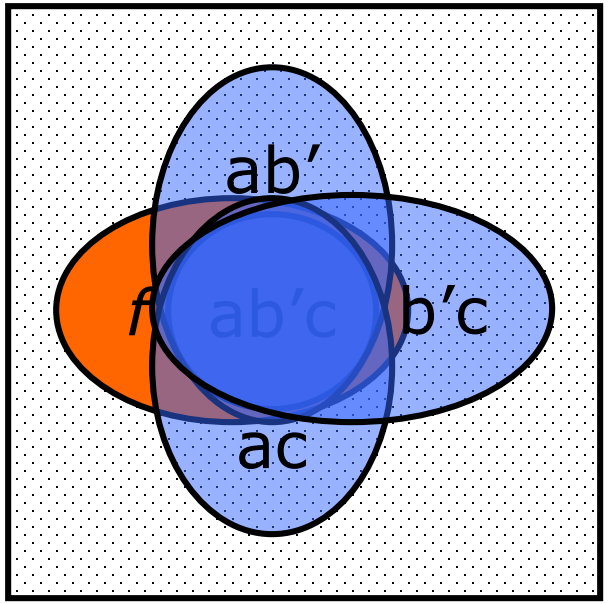
- Removing a literal from any product term (any implicant) makes it cover twice as many minterms.
  - Removing a literal "grows" the term
  - ex. 3 variables:



ab'c covers 1 minterm

ab'
ac
b'c

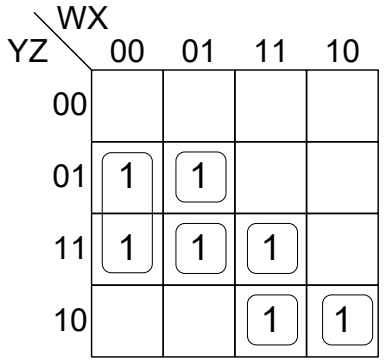
} each cover 2 minterms



ab'c is an implicant of *f*.

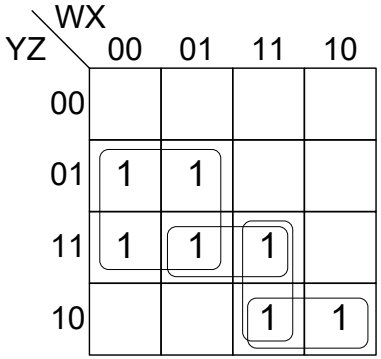
Any way of removing a literal makes ab'c no longer imply *f*. So ab'c is a **prime implicant** of *f*.

Implicant: Any product term that *implies* a function F (i.e., if, for some input combination, product term P = 1, then F = 1 for the same input combination)

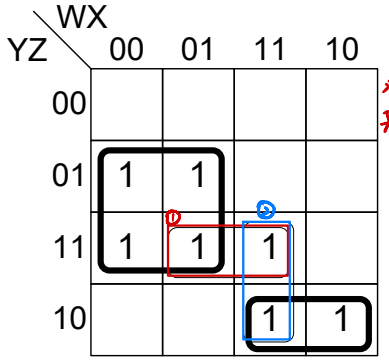


These are only a few of the implicants of this function…

Prime Implicant: An implicant such that if one literal is removed, the resulting product term no longer implies F



Essential Prime Implicant: A prime implicant that covers a minterm that is not covered by any other prime implicants



对于这个落单的，我们有两种方式来cover它.

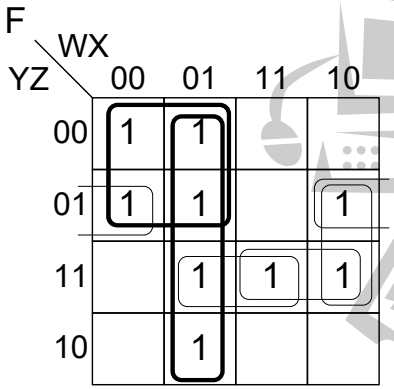
by idenpotency I can cover one minterm several times, but I don't have to.

Theorem: The minimal SOP of a function is a sum of prime implicants

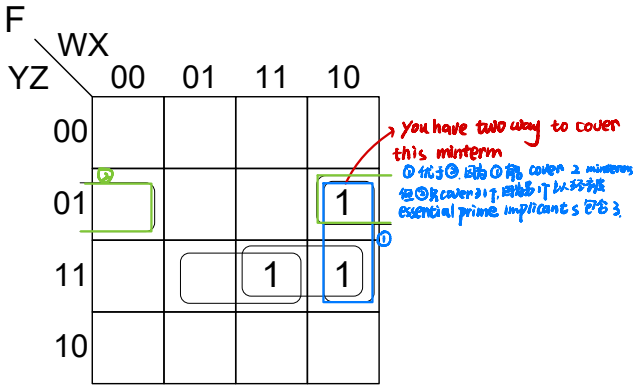
K-map minimization procedure:

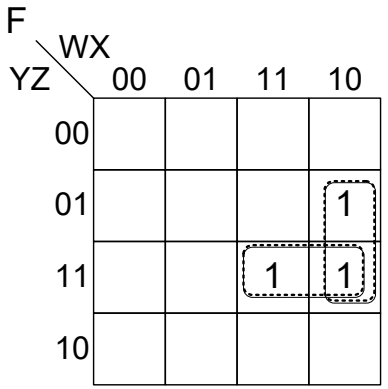
1) Find all prime implicants (PIs)
2) Determine which PIs are essential prime implicants (EPIs) and include those in the minimal SOP. *Remove all EPIs and the 1-cells they cover*
3) Remove all PIs that are **eclipsed (dominated)**
   - Prime implicant X *eclipses* prime implicant Y if X covers at least all of the 1-cells that Y covers
4) Find secondary EPIs, include secondary-EPIs in minimal SOP, and remove 1-cells they cover. If there are any 1-cells remaining, goto step 3

- *Note: If, at any stage, there are no remaining EPIs, all remaining combinations of PIs must be considered for the minimal SOP*



$$F = \overline{W}\,\overline{Y} + \overline{W}X$$





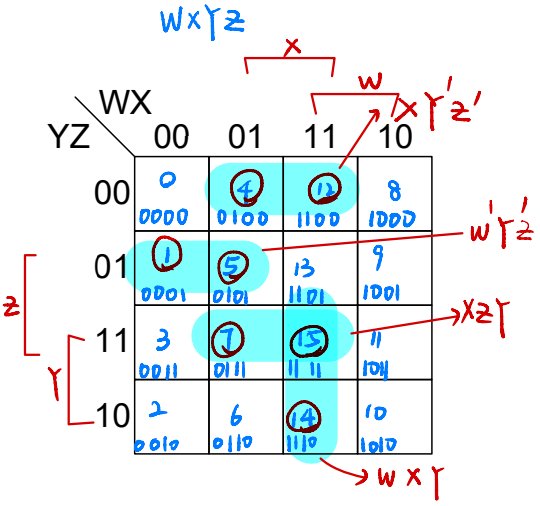
$$F = \overline{W}\,\overline{Y} + \overline{W}X + WYZ + W\overline{X}Z$$

cost = 2 + 2 + 3 + 3 + 4x1 = 14

In Class Exercise

Find the minimal SOP for the following function:

$$F = \sum_{(W,X,Y,Z)} (1,4,5,7,12,14,15)$$

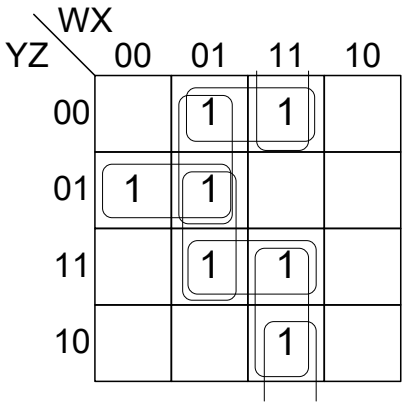


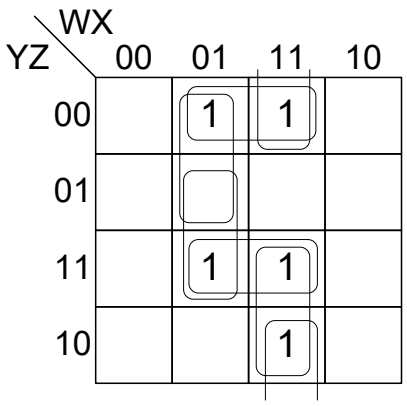
$F = x'z' + w'z' + xz + wx'$

In Class Exercise

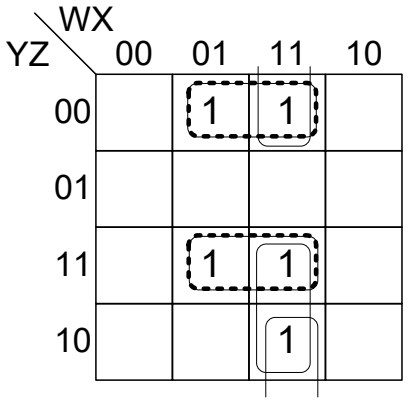
Find the minimal SOP for the following function:

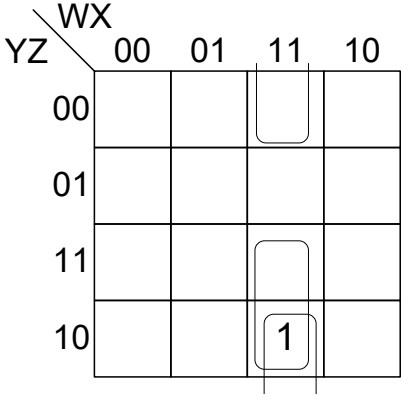
$$F = \sum_{(W,X,Y,Z)} (1,4,5,7,12,14,15)$$





$$F = \overline{W}\,\overline{Y}Z$$





$$F = \overline{W}\,\overline{Y}Z + X\overline{Y}\,\overline{Z} + XYZ$$

$$F = \overline{W}\,\overline{Y}Z + X\overline{Y}\,\overline{Z} + XYZ + WXY$$

or

$$F = \overline{W}\,\overline{Y}Z + X\overline{Y}\,\overline{Z} + XYZ + WX\overline{Z}$$

The minimal SOP may not always be unique!

Using K-maps to find minimal POS

根据"1"的分布位置，有时minterm如，有时Maxterm如。

1) Find minimal SOP for $\overline{F}$ 与其找 maxterm，不如0
2) Use DeMorgan's to get minimal POS for F

$$F = \sum_{W,X,Y,Z}(1,3,4,6,9,11,12)$$

F

YZ \ WX	00	01	11	10
00		1	1	
01	1			1
11	1			1
10		1		

$\overline{F}$

YZ \ WX	00	01	11	10
00	1			1
01		1	1	
11		1	1	
10	1		1	1

X, W, Z, Y

$$\overline{F} = \overline{X}\overline{Z} + XZ + WY\overline{Z}$$

$$F = (X + Z)(\overline{X} + \overline{Z})(\overline{W} + \overline{Y} + Z)$$

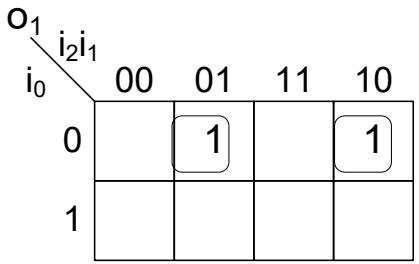
count cost: 2+2+3+3 = 10

Don’t Care Outputs

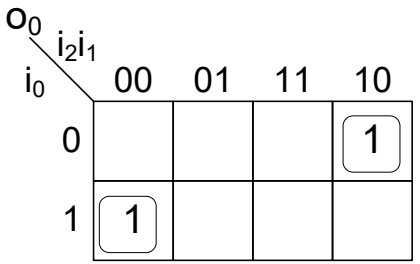
- Sometimes when designing a function, some input combinations may never occur
- Design Example - *Encoder*: Converts 3-bit one-hot code to equivalent binary code

i_2	i_1	i_0	o_1	o_0
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
1	0	0	1	1
others			0 (d)	0 (d)

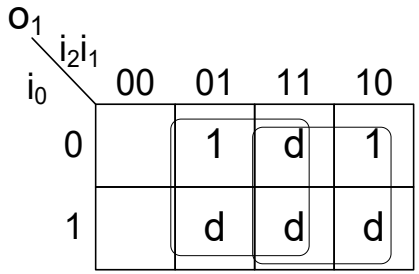
How to assign unused input combinations?



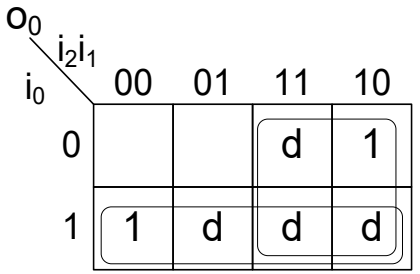
6 literals



6 literals



2 literals

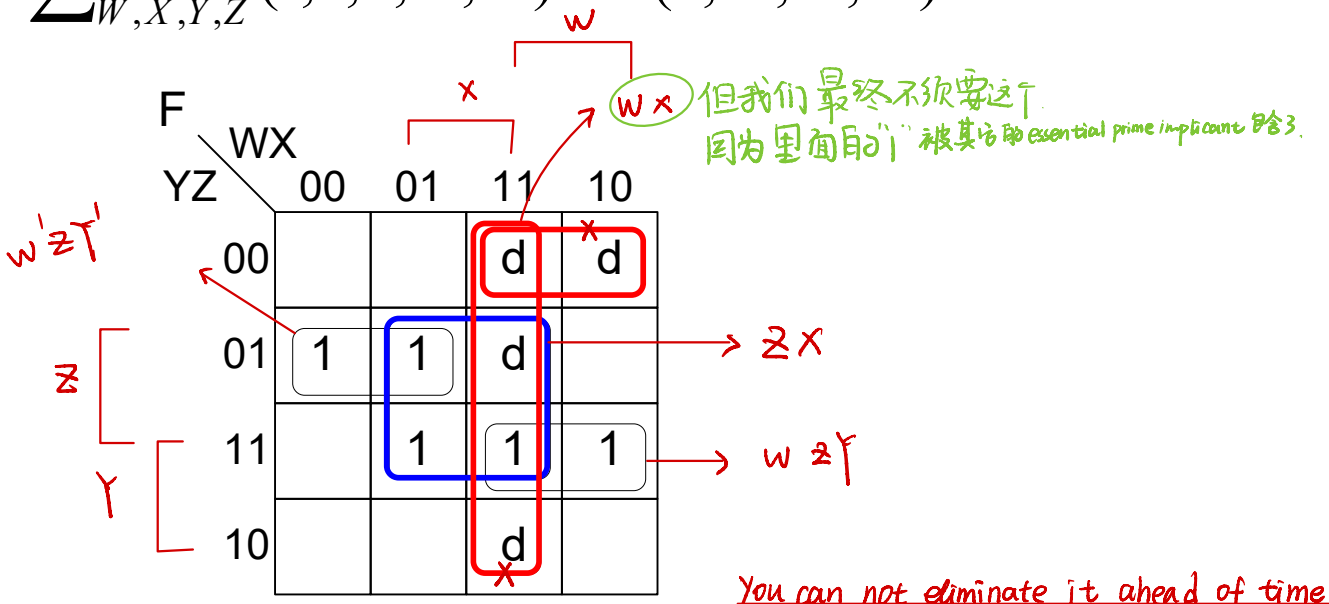


2 literals

Don’t cares allow us to easily assign 1 or 0 to unused input combinations in order to minimize implementation cost!

- When finding minimal SOP with don't cares:
  - Use *d* cells to make PIs as large as possible
  - No prime implicant should include only *d*'s
  - Only 1-cells should be considered when finding minimal covering set of PIs

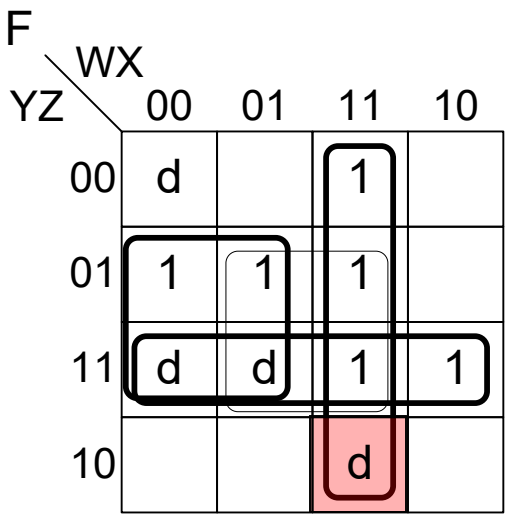
$$F = \sum_{W,X,Y,Z}(1,5,7,11,15) + d(8,12,13,14)$$



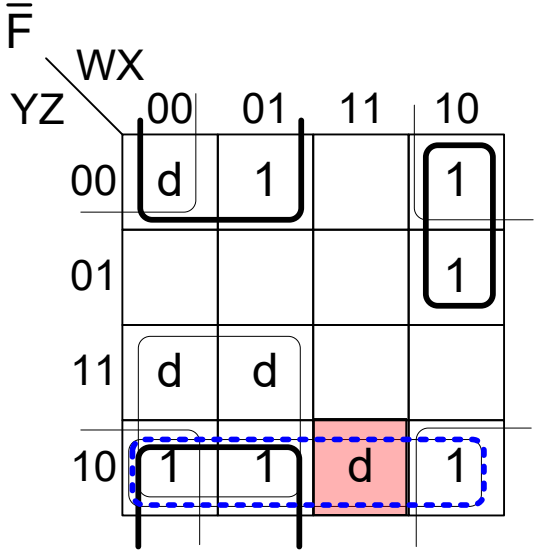
$$F = XZ + \overline{W}\,\overline{Y}Z + WYZ + W\overline{X}$$

You can not eliminate it ahead of time.
Because you have to actually identify all the prime implicants before you start minimization.
So the minimal solution must be a subset of prime implicates
Most of time. Some are essential prime implicant. the rests are covered by them

Minimal SOP vs. Minimal POS w/ Don't Cares



$$F_{SOP} = WX + YZ + \overline{W}Z$$



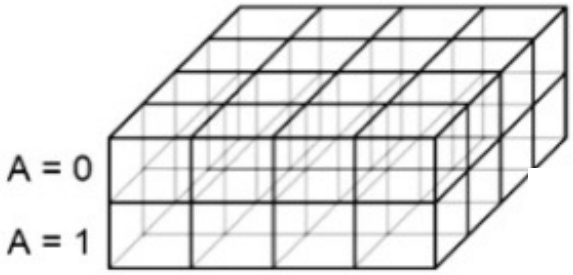
$$\overline{F} = \overline{W}\,\overline{Z} + Y\overline{Z} + W\overline{X}\,\overline{Y}$$

$$F_{POS} = (W + Z)(\overline{Y} + Z)(\overline{W} + X + Y)$$

- F_{SOP} has a smaller literal cost than F_{POS}
- WXYZ = 1110: $F_{SOP}(1,1,1,0) = 1$, $F_{POS}(1,1,1,0) = 0 \rightarrow F_{SOP} \neq F_{POS}$!!!
- When don't cares are involved, in general the minimal SOP for a function is *not equal* to the minimal POS for the function
  - But that's ok! We don't care about the input combinations where they differ…

5 Variable K-Maps

- Every cell is adjacent to *5* other cells



F

DE \ BC	00	01	11	10
00	m_0	m_4	m_{12}	m_8
01	m_1	m_5	m_{13}	m_9
11	m_3	m_7	m_{15}	m_{11}
10	m_2	m_6	m_{14}	m_{10}

A = 0

F

DE \ BC	00	01	11	10
00	m_{16}	m_{20}	m_{28}	m_{24}
01	m_{17}	m_{21}	m_{29}	m_{25}
11	m_{19}	m_{23}	m_{31}	m_{27}
10	m_{18}	m_{22}	m_{30}	m_{26}

A = 1

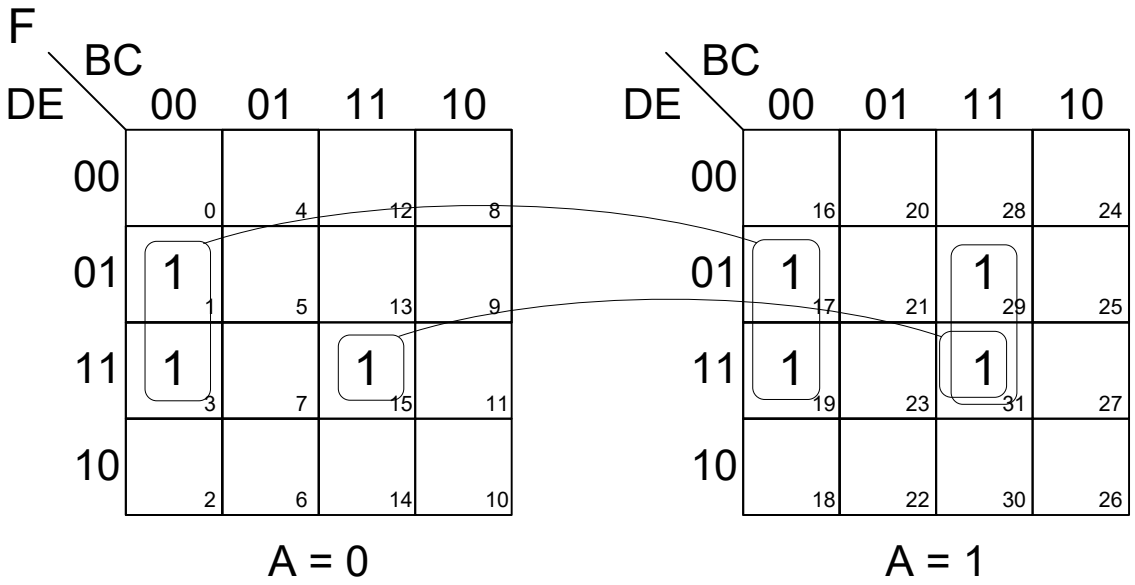
Keep life simple by splitting on the MSB!

Q: How many prime implicants are possibly there for a function?

A: the #square in the space $= 2^n$,

the prime implication is a product term (is an "and" of up to n literals) 对于每个位置有三种情况: complement / uncomplement / missing 3^n

$$F = \sum_{A,B,C,D,E}(1,3,15,17,19,29,31)$$



$$F = \overline{B}\,\overline{C}E + BCDE + ABCE$$

$$F = \sum_{A,B,C,D,E}(1,3,15,17,19,29,31)$$

BEQ

File Edit Zoom Examples Options Help

Boolean Minimizer

Input Window

```
A B C D E F
0 0 0 0 1 1
0 0 0 1 1 1
0 1 1 1 1 1
1 0 0 0 1 1
1 0 0 1 1 1
1 1 1 0 1 1
1 1 1 1 1 1;
```

Minimize

Go

Output Data

Search

Output Options

- [ ] Comment Lines
- [ ] Input Eqn / TT
- [ ] Truth Table - Full
- [x] Minimize
- [ ] Invert & Minimize
- [ ] Align Minterms
- [x] PLA Truth Table
- [ ] Sort Terms
- [ ] Logic Design

0 Operator Format

```
10/3/2021
MINIMIZE TRUTH-TABLE

(1)  * 25 unspecified truth table rows

        F = not B and not C and E or A and B and C and E or B and C and D and E;

.i 5
.o 1
.ilb A B C D E
.ob F
.p 3
-00-1 1
111-1 1
-1111 1
.e
```

Ln 8 Col 13 Overwrite Ln 1 Col 1